STAG: Smart Tools and Applications in Graphics (2022)
D. Cabiddu, T. Schneider, G. Cherchi, and R. Scateni (Editors)

Outside-in priority-based approximation of 3D models in LEGO bricks

Filippo A. Fanni[1], Elisa De Rossi[1] and Andrea Giachetti[1]

[1]Department of Computer Science, University of Verona, Italy

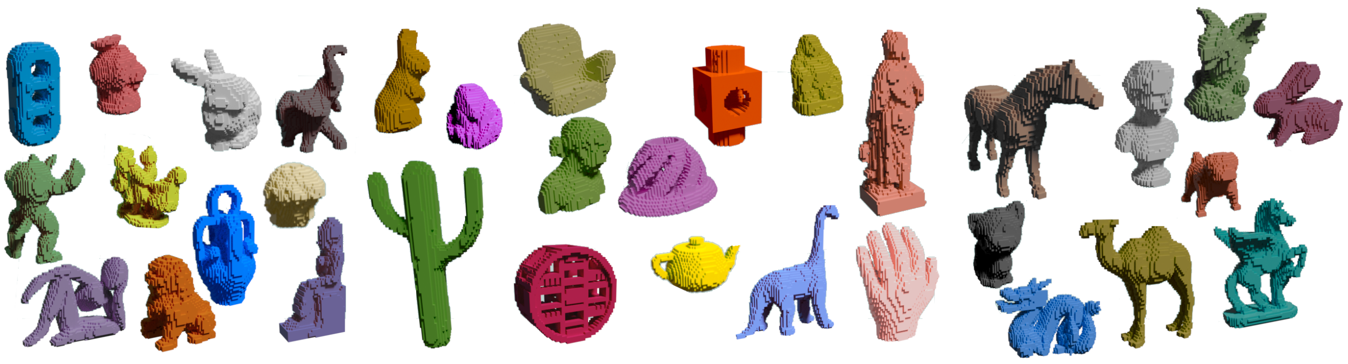

Figure 1: *The 33 models of our BRICKS benchmark at the intermediate of the three resolutions provided.*

Abstract

In this paper, we discuss the problem of converting a 3D mesh into an assembly of LEGO blocks. The major challenge of this task is how to aggregate the voxels derived by the shape discretization into a set of standard bricks guaranteeing global connectivity.

We propose an outside-in priority-based heuristic method based on the analysis of the critical regions that are more likely to cause the creation of a legal assembly to fail. We show that our graph-building heuristic provides relevant advantages, making it easier to obtain a connected graph with good properties with respect to the layer-based or random aggregation strategies applied in most of the optimization approaches.

We also propose BRICKS, a novel dataset for the evaluation of aggregation strategies. It includes voxelizations at 3 different resolutions of 33 shapes and allows the easy comparison of different voxel aggregation strategies independently of the shape discretization step and also considering their scalability. We use it to evaluate our approach with respect to graph-based connectivity measures, showing the advantages of the proposed strategy.

CCS Concepts

• ***Computing methodologies*** → ***Shape modeling; Mesh models; Mesh geometry models;*** *Volumetric models;*

1. Introduction

The idea of automatically decomposing a 3D shape into a buildable assembly of standard LEGO bricks has been proposed in several research papers [TSP13; LYH*15; LKKM15; LKM18; FDSG21; KS21]. The problem is typically stated as follows: given the voxelization of a 3D model, initialize (1x1) bricks in the voxels' positions, then replace these bricks with larger ones from a standard dictionary to obtain a unique and robust connected component. Many published works show impressive results, providing the automatic generation of tightly connected and stable assemblies and sometimes color fidelity to the original model for various input meshes.

However, all the methods are based on heuristic optimization and are hard to evaluate comparatively. They are typically tested on small proprietary data, including the voxelization step in the pipeline and optimizing the layout with respect to different sets of desiderata (connectivity, stability, color, etc.).

In our work, we propose two specific contributions to this field,

© 2022 The Author(s)
Eurographics Proceedings © 2022 The Eurographics Association.
This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

delivered by EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

DOI: 10.2312/stag.20221256

focusing on the voxel aggregation step only with the simple goal of providing connected assemblies with a limited number of bricks.

First, we propose a new outside-in priority-based heuristic approach that focuses on merging the bricks that are harder to connect first. Most approaches are based on random or per-layer initialization of the LEGO bricks' layout, not considering existing priors on the location of critical regions where disconnected components are likely to be created. We observe that the problems are mostly related to the difficult connection of the external bricks not attached to the upper of the lower layer, and we show that, by using our priority-based approach, it is possible to improve the quality of the connectivity-based optimization. Second, we propose *BRICKS*, a dataset of voxelized shapes that allows the comparison of the aggregation approaches independently on the voxelization step. The dataset is derived by meshes with different topology and geometrical complexity, and includes voxelizations at three different resolutions to test the scalability of the aggregation methods. We use BRICKS to evaluate the improvements obtained on different connectivity metrics using the proposed approach and also to compare our results with those provided by the publicly available implementation of the approach proposed in [TSP13].

Both the BRICKS benchmark and our implementations of the brick layout generation algorithms discussed in the paper are available at the link `https://univr-vips.github.io/bricks/`.

2. Related Work

Many research works have been dedicated to the development of methods to build 3D models with assemblies of rigid parts. A recent survey [WSP21] discusses these works in general. Here we discuss the scientific papers focused on the approximation of shapes with standard LEGO bricks.

The first work on this specific topic is probably the one by Gower et al. [GHP98], where a local cost function maximizing bricks attachment and stability was proposed. The work suggests the evaluation of the bricks overlaps in adjacent layers but does not propose a practical algorithm.

A complete method was proposed by Testuz et al. [TSP13]. The algorithm creates the first layout by iteratively selecting random bricks and aggregating them with the neighbors favoring the merging that create the most connections. A subsequent graph-based optimization selects borders connecting connected components and weak connections and tries to reconfigure locally the layout.

The method proposed in [LYH*15] also starts from an iterative aggregation of bricks where the next 1x1 brick to be merged into a larger one is chosen randomly. Also in these methods, a set of weak elements is found with graph analysis and an iterative procedure selects randomly a k-ring neighborhood of a random element and performs multiple reconfigurations of this region trying to maximize the cost function. In this work color and stability-based optimization were also considered.

Lee et al. [LKKM15] propose a genetic algorithm creating mutations of initial configurations, based on layer processing. The initial population is created also in this case with random aggregation, iteratively selecting original voxels and extending them locally. The genetic algorithm is then used to generate mutations and merge and select the solution minimizing the cost.

In a later work [LKM18], the same authors adopt a similar random initialization method with a per-layer approach, optimizing layers based on a cost depending on the number of bricks in the current layer, the number of bricks in the lower layer that is connected to each brick and the number of bricks that cover the lower layer perpendicularly.

A more deterministic initialization of the solution has been recently proposed in [Ste16; KS21]. Stephenson [Ste16] used several heuristics to build layers in sequence from bottom to top. The method searches to minimize, in order of importance, the number of connected components in the model, the number of undesirable edges in the model, and the number of bricks in the model.

The method of Kollsker et al. [KS21] also works in subsequent layers generating 1D strips with alternate directions and merging them with specified heuristics. Finally, strips are filled with bricks, here locally optimizing a cost function. A subsequent destroy and repair procedure is further applied to improve the assembly's properties.

Many of these works present nice results on a limited number of models. However, all the proposed approaches are based on heuristics, and greedy procedures and often depend on layer order. It is clear that the problem is not solvable in general and global optimization is not feasible, but simple deterministic strategies could in principle improve the efficiency of the algorithms and the quality of the results.

The results, on the other hand, cannot be easily compared as a benchmark to evaluate them is missing.

The largest dataset used to test the brick aggregation approaches has been used in [Ste16] where 30 models have been used, and interesting outcomes on model features and non-buildable parts have been discussed. However, the voxelizations are not publicly available for comparison, and single resolutions have been employed.

For these reasons, we propose two contributions to the research on this topic, namely:

- a novel outside-in priority-based aggregation procedure that provides improved layout optimization with reduced complexity
- a dataset of voxelized models that can be used to compare different brick layout generation approaches

3. Outside-in priority-based legolization

In our work we consider the legolization problem starting from an already voxelized shape and considering the possible aggregation of the smallest 1x1 bricks into a set of legal bricks of sizes 1x2,1x3,1x4,1x6,1x8,2x2,2x3,2x4,2x6,2x8.

A simple observation about the connection of LEGO blocks in an assembly trying to fill the original model volume is that the major issues are related to the external parts. It is near the border where we can have 1x1 blocks/voxels that are not attached to the upper and lower layers and we need to ensure that they are merged with connected, inner blocks.

A reasonable strategy to avoid these issues is to start the aggregation of small bricks into larger ones from the borders, trying to ensure that external parts are connected to the internal ones. We decided to implement a strategy of this kind, starting with the aggregation from the outer and the more problematic elements, e.g. those not connected with the upper and lower layers. Different from the other approaches trying to optimize the layout while creating the initial aggregation (e.g. [Ste16; KS21]) we don't want to use a layer-by-layer procedure, but rather adopt a global approach.

In our formulation the LEGO assembly is characterized by two graphs: one is the *attachment graph*, describing the connections between bricks created by studs, and the other is the *layer adjacency graph*, where the arcs join the bricks that have a contact surface on a layer. The first graph determines the connectivity of the assembly, and the second the potential couples of bricks that could be merged.

When a voxelization is created and filled with 1x1 blocks, the *attachment graph* links bricks with the corresponding ones in the upper and lower layers (if they exist), the *layer adjacency graph* links with the existing elements in the 4-voxels neighborhood of the same layer. The idea of our method is to use the graphs not only to define a cost function to make the attachment graph more connected but also to define a priority queue selecting the bricks to be merged in a deterministic way.

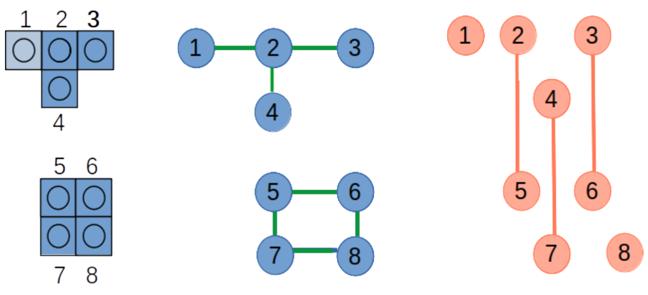

Figure 2: *Let's consider, for example, an initial two-layer voxelization with a top layer with 4 1x1 bricks (1-4) and a bottom layer with 4 1x1 bricks (5-8) supporting only part of the upper one. The blue node-arc plot (middle column) represents the corresponding adjacency graph, while the orange plot represents the corresponding attachment graph*

Figure 2 shows an example of simple voxelization. The initial layers/bricks are shown in the left column, the *adjacency graph* in the middle column and *attachment graph* in the right one. The aggregation of the bricks in larger ones should start from a couple of neighboring bricks, e.g. from an arc of the adjacency graph. Let us consider the upper layer: three pairs of bricks could be merged (1-2,2-3,2-4). However, if we start merging a random pair, we would probably result in a final disconnected result, as it happens in two of the three cases (2-3,2-4). The critical point is the connection of brick 1 which is not linked in the attachment graph and has just a single connection in the adjacency graph.

Our solution is to adopt a priority-based aggregation, based on the local connectivity of the two graphs. We manage the priority queue globally, avoiding biases related to the layer-based aggregation methods (proposed, for example in [LKM18; KS21]).

The two graphs are initialized from the 1x1x1 bricks, as in the example of Figure 2. Each brick is assigned a priority value of:

$$\rho(n) = \omega_1 * I(n) + \omega_2 * \deg(n, G_{at}) + \omega_3 * \deg(n, G_{adj}) \quad (1)$$

where $I(n)$ is a function that determines whether the node n is on the border of the layout, $\deg(n, G_{at})$ and $\deg(n, G_{adj})$ are, respectively, the degrees of the node n in the attachment and adjacency graph. Furthermore, the weights $\omega_1, \omega_2, \omega_3$ are chosen so that the latter terms are considered only in the case of equality among the previous ones. The same weights were used in all the results reported in this paper.

This priority is exploited in the iterative algorithm that merges the smaller blocks into larger ones (Algorithm 1). The algorithm tries to merge the existing bricks with neighbors, trying first to merge them with 1xN bricks of increasing length and then with 2xN bricks of increasing length.

For each of these increasing sizes, the aggregation is based on a priority queue determined by the values $\rho(n)$ assigned with Equation 1. As there can be many bricks with the same value, every time we generate the queue we add small random perturbations to remove biases in the aggregation behavior.

After the node to process is selected, the algorithm looks for the best neighbor to merge it with in the adjacency graph. If there is more than one neighbor that can be merged creating a valid brick, the preference is given to the one with the highest value of $\rho(n)$. If multiple nodes have the same priority, the preference is based on the preferred direction associated with each layer. In fact, given the initial voxelization, we associate a preferential direction to each layer, alternating between adjacent layers (e.g. odd layers: vertical, even layers: horizontal). This is done by starting from the layer with the maximal footprint and associating to it the direction of maximal elongation, and then iteratively associating perpendicular directions to the neighboring layers to facilitate the orthogonal crossing of bricks. Lastly, if multiple bricks respect the directionality constraint, or if none does, a random selection is performed.

To compare our priority-based solution with the classical random aggregation of bricks, we implemented a different version of the code where the priorities are randomly generated. This should make the initial aggregation results similar to those obtained by the methods of [LYH*15; LKM18].

The fact that the priority-based solution provides better results is clear just by looking at the toy examples in Figures 3 and 4.

However, the methods based on random aggregations usually perform multiple initializations and then refine the initial results by iteratively reassembling the weaker parts trying to optimize a local cost function maximizing the connectivity. We, therefore, implemented an iterative optimization procedure of this kind to evaluate the effects of our approach in the complete framework.

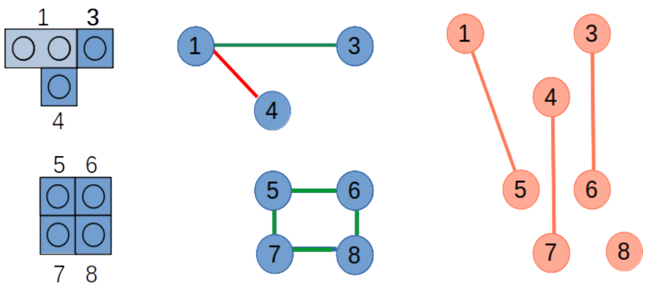

Figure 3: *After the first iteration of the priority-based selection on the initial configuration in Figure 1, brick 1 is selected for its highest priority and merged with brick 2. The adjacency graph has now two links in the top row, and only one (green) can result in a legal merging, while the red link represents a merging that would result in a non-existing LEGO brick.*

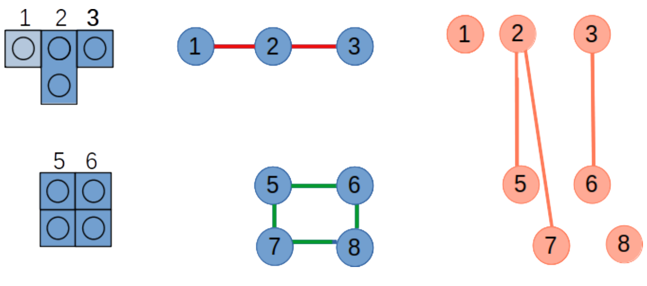

Figure 4: *A random selection of the first brick to merge (here 2 with 4 may lead to a non-connected and not mergeable configuration of the top layer (any merge would create an L-shaped brick not included in the dictionary)*

3.1. Iterative reassembly of weak parts

The method we use to perform an iterative reassembly of selected weak parts is similar to the one used in [LYH*15] and is shown in Algorithm 2. We focus on the reassembly of the layout wherever multiple components form in the attachment graph, i.e. a subset of LEGO bricks are not attached to the rest of the construction. From our experiments, this always happens on the outer shell of the constructions, and these components are typically composed of a small number of bricks.

An important aspect to consider is that there is no guarantee that a connected layout exists, and thus no guarantee of convergence can be given in this step. To prevent infinite loops, we limit the number of iterations done at this step up to the number of the connected components plus an additional constant (20 was used for all the results presented in the paper).

The algorithm repeatedly selects one of the connected components, except the biggest one that is assumed to be the main body of the construction, places a window of size $X \times Y \times Z$, replaces all the bricks that intersect this window with 1x1 bricks, and repeats the Algorithm 1.

To reduce the probability of the algorithm looping over the same connected components, generating the same layout and thus never converging, we introduce a few random factors. The dimensions X,

Algorithm 1 Brick merging

```
procedure BUILD_LAYOUT($G_{at}, G_{adj}$)
    for t_shape ∈ [1x1, ⋯, 1x4, 1x6, 1x8, 2x2, 2x3, 2x4] do
        pq ← INIT_PRIORITY_QUEUE($G_{at}, G_{adj}$)
        while pq $\neq \varnothing$ do
            $\nu$ ← SELECT_NODE(pq)
            $N$ ← NEIGHBORS($\nu$)
            for $f \in$ [feasibility, priority, direction, random] do
                $N$ ← FILTER($N, f$)
                if $N = \varnothing$ then
                    break
                else if $N = \{n\}$ then
                    $\nu'$ ← MERGE_NODES($\nu, n$)
                    REMOVE_NODE($n$, pq)
                    INSERT_NODE($\nu'$, pq)
                    break
                end if
            end for
        end while
    end for
end procedure
```

Algorithm 2 Connected component merging

```
procedure MERGE_COMPONENTS($G_{at}, G_{adj}$)
    $\tau \leftarrow 0$
    while $\tau < \tau_M$ and |COMPONENTS($G_{at}$)| > 1 do
        $\tau$++
        $C$ ← RANDOM_COMPONENT($G_{at}$)    ▷ This never yields
the biggest component
        $b$ ← RANDOM_PICK(
          $\{\beta | \beta \in C \wedge \exists \alpha : (\beta, \alpha) \in G_{adj} \wedge \alpha \notin C\}$)
        $B$ ← NEARBY_BRICKS($b$, RANDOM_SIZED_WINDOW)
        DISASSEMBLE_GRAPHS($B, G_{at}, G_{adj}$)
        BUILD_LAYOUT($G_{at}, G_{adj}$)
    end while
end procedure
```

Y, and Z of the window are chosen randomly among the set $\{5,7\}$ for X and Y, and between $\{3,5\}$ for Z. Additionally, the center is placed randomly over one of the bricks comprising the component that is also adjacent to one brick of another connected component. Lastly, the selection of the connected component to be processed is also random.

4. Evaluation dataset: BRICKS

As we discussed before, previous works are tested on a limited number of models, and every work proposes its own models and evaluation functions. While this can be beneficial to evaluate very specific use cases, we argue that a more standard approach to both the dataset and the objectives would be highly beneficial and improve the ability to compare different approaches when the implementation is not publicly released.

Our goal is to evaluate how the methods implemented for the block aggregation behave on the same voxelization, measuring the connectedness of the resulting block graph, and the complexity of the method scales with the growing number of voxels.

We therefore created BRICKS (`https://univr-vips.github.io/bricks/`), a public dataset composed of *voxelized* models at *three different resolutions* that roughly correspond to reasonable scales of model reproduction.

The dataset is built from 33 shapes, of which we provide the three dense matrix representations for the voxelizations, the three dense matrices of only the voxels intersecting some mesh's triangle, and an appropriately scaled mesh for each resolution so that the mesh scale matches the lego structure when the voxels are converted to bricks with standard dimensions (8x8x9.6 millimeters). A .csv file containing some statistics of the voxelizations is also provided.

The models are a hand-selected subset of the dataset from [PNA*21]. After the selection, we remeshed all the models that were not manifold, and then manually rotated them to have the expected z axis direction for practical building.

The orientation along the brick x and y directions (e.g. voxelization x and y directions) are chosen with an optimization criterion, so as to possibly align the flat parts of the objects to the directions of the bricks. For this goal, we employ a voting approach. Each triangle of the mesh casts a weighted vote for a specific angle formed by the projection of its normal on the xy plane and the x-axis. The projected normal also defines a plane, perpendicular to the xy plane by construction, and the projection of the triangle's area onto this plane is taken as the weight of the vote. In this way, we weigh more the votes of large vertical triangles and decrease the importance of almost horizontal ones. We use a kernel density estimation to extract a continuous distribution from the votes, taking care of the cyclic behavior at the boundaries. Finally, we extract the maximum of the function and use it to align the mesh along the x-axis of the discretization grid with a simple rotation.

To define LEGO models of different sizes, we finally scale the meshes to three different target volumes, so that, using the standard voxel grid with elements of 8x8x9.6 millimeters, we generate three different voxelizations with a different number of elements. The target volumes are .7, 7, and 70 liters, that roughly correspond to an average of 2k, 15k, and 130k voxels for the low, medium, and high resolution respectively.

Examples of selected meshes and related voxelizations are shown in Figure 5. All the voxelizations at the intermediate resolution are represented in Figure 1.

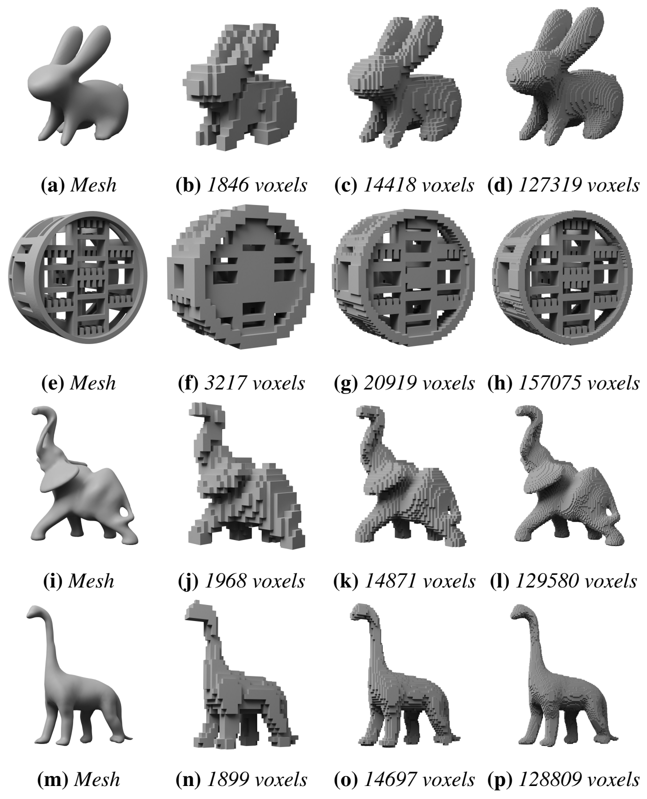
(a) *Mesh* **(b)** *1846 voxels* **(c)** *14418 voxels* **(d)** *127319 voxels*
(e) *Mesh* **(f)** *3217 voxels* **(g)** *20919 voxels* **(h)** *157075 voxels*
(i) *Mesh* **(j)** *1968 voxels* **(k)** *14871 voxels* **(l)** *129580 voxels*
(m) *Mesh* **(n)** *1899 voxels* **(o)** *14697 voxels* **(p)** *128809 voxels*

Figure 5: *Examples of original mesh and the related voxelizations at three different resolutions with the 1x1 brick x/z ratio.*

5. Results

Using BRICKS, we try to assess the effectiveness of the Outside-in Priority-based (OP) aggregation in providing a better initialization of the elements' layout. We hypothesize that it could lead not only to a better initial layout but also to faster convergence to a connected and buildable assembly.

The metrics used to assess the quality of the assembly are, for the tests presented in this paper, only based on the number of the connected components created, the number of bricks, and the node connectivity of the attachment graphs. As the computational complexity of the node connectivity estimation is high, we considered a *sampled node connectivity* selecting, for each voxelization, 10 pairs of voxels and estimating the average node connectivity on these pairs only. A proper measure would require testing all the pairs of nodes in the graph and then averaging the results, but this is computationally unfeasible for our graphs

In this work, we don't use optimization methods based on forces or stability and therefore we don't evaluate related values. We plan to add such methods and the related evaluation metrics in future work, as they can be added without changing the models included in BRICKS.

Here we first compare the quality and the computational cost of the initial layouts obtained with a repeated random aggregation and the OP-based layouts and then we show how the results are improved with the iterative reassembly.

5.1. Layout initialization, random vs OP

For all the 99 voxelizations of BRICKS, we performed multiple runs of the random and OP-based aggregation initialization. The results are shown in Table 1. In detail, the table reports the averages on the 33 models at the different resolutions of the average computation times of the repeated runs, the average number of connected components obtained and the average number of the

© 2022 The Author(s)
Eurographics Proceedings © 2022 The Eurographics Association.

connected component in the best of the repeated runs (where the best result is defined as the one with the lowest number of connected components and blocks), the average number of bricks in the different runs and the average number of bricks in the best run of each model, the Sampled Node Connectivity (SNC) averaged for all the runs and for the best runs, and the percentages of shapes where the initialization step obtains a single connected component (% success).

It is possible to see that on average, the results of a run of the OP aggregation results in a lower number of connected components and bricks.

Regarding the bricks' number, the averages are, respectively 12972 and 9484, with an average improvement of 24% with OP.

The average number of connected components obtained with random and OP aggregations are 7.6 and 1.7 respectively, with an average improvement of 64% with OP.

The box plots in Figure 6 show the distributions of numbers of connected components obtained with the random and OP approaches. The difference is large and becomes larger with the number of blocks. Furthermore, as shown in Figure 7 even comparing the number of connected components of the best layout generated by the two methods, reveals a pronounced superiority of the OP method

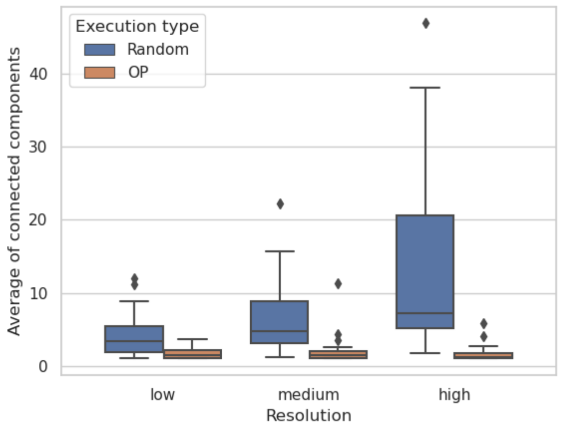

Figure 6: *A box plot representing the number of connected components generated by the two methods.*

A further justification for the use of the OP approach can be obtained by analyzing the characteristics of the connected components obtained by the simple runs of the aggregation approaches. Table 2 shows the statistics of these secondary components for all the runs performed with the two methods. It turns out that the results are always composed of a large principal component and a set of very small detached ones all lying at the shape borders. In Figure 8 we show an example of construction generated with the random approaches with the detached components highlighted in different colors.

Interestingly, while the OP method produces on average fewer

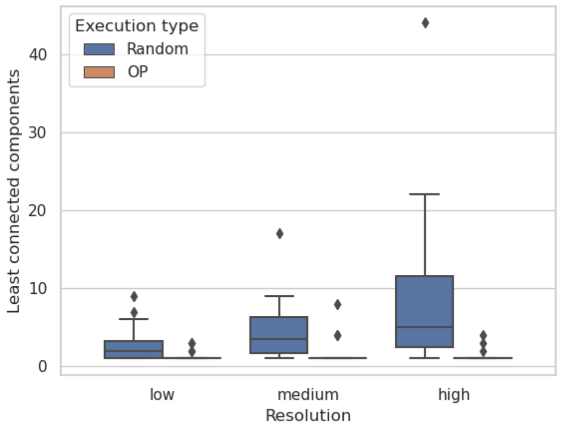

Figure 7: *A box plot representing the least number of connected components generated by the two methods for each model over the course of 10 runs.*

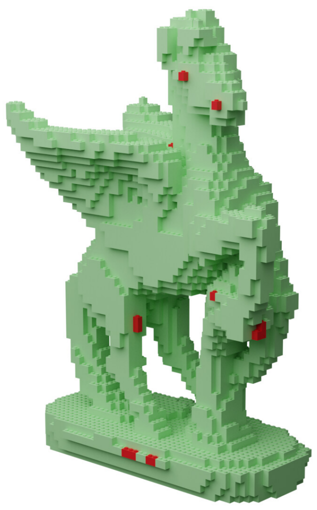

Figure 8: *Rendering of a brick assembly created from the medium-resolution Pegaso voxelization. There are many small connected components (on average 16 in our tests with this model) not attached to the main one, here highlighted in red.*

components, the size of these components is slightly larger. This is because the OP method starts from the outside, thus creating larger external bricks that, when disconnected, create larger components than the ones generated from the random approach. However, given how small the components are in both cases, their dimensions become an irrelevant factor against their number, which instead determines how many merging iterations will be necessary, and how much time will be spent on fixing them.

The box plots in Figure 9 show the distributions of the sum of the number of voxels in the secondary components generated by the random and OP approaches. Similar to the number of connected components, their differences become more pronounced as the res-

	Method	Time	Avg.bricks	Best bricks	Avg.CC	Best CC	Avg. SNC	Best SNC	% success
Lo-res	random	0.45 s	542.6	531.8	4.1	3.5	2.96	**3.53**	25.7%
	OP	**0.39 s**	**446.6**	441.1	**1.7**	**1.3**	**2.69**	3.09	**62.8%**
Mid-res	random	4.76 s	4039.2	3983.1	6.6	4.2	3.61	**3.63**	13.3%
	OP	**4.08 s**	**2982.0**	2957.9	**1.9**	**1.4**	**3.69**	3.47	**61.9%**
Hi-res	random	141.6 s	34335.5	33853.0	12.3	9.3	4.11	3.88	3.0%
	OP	**114.8 s**	**25023.8**	24925.8	**1.6**	**1.3**	**4.74**	**4.74**	**66.0%**

Table 1: *Averages grouped by resolution and method, of the aggregation metrics estimated on 10 repeated runs performed on all the voxelizations included in the BRICKS dataset.*

	Execution type	Bricks per component	Voxels per component	Outside percentage
Lo-res	Random	1.7	4.1	100%
	OP	1.2	3.4	100%
Medium	Random	1.5	3.1	100%
	OP	3.0	10.6	100%
High	Random	1.4	2.3	100%
	OP	2.5	9.7	100%

Table 2: *For each resolution and algorithm we report the average of the number of bricks and voxels in the non-maximal connected components, and the percentage of components with at least one voxel on the surface of the model.*

olution increases. Figure 10 shows instead the average number of voxels in the secondary components. The OP approach tends to produce few enough that the boxes are flattened at 0.

Interestingly, both Figure 9 and 10 show an evident outlier in the OP method, that reaches 266 voxels total among 7 secondary connected components. We show a result obtained on this model in Figure 11. We can see how the OP method fails at connecting the thin bridges present in the model to the main body, resulting in several connected components. However, the iterative reassembly discussed in Section 5.2 can generate a connected layout even starting from these configurations.

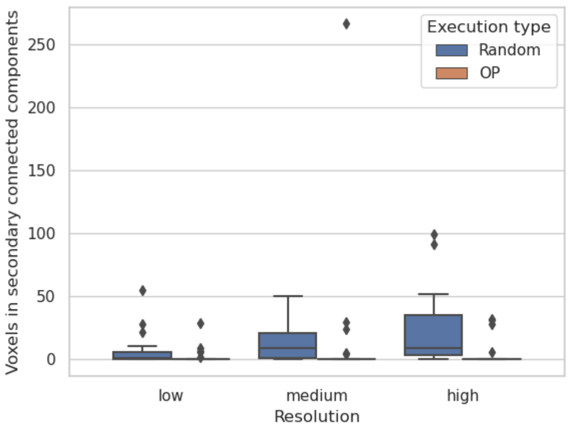

Figure 9: *A box plot representing the total number of voxels in the secondary connected components of the best layouts generated over the course of 10 runs.*

Another interesting fact is that even the OP approach not only creates better initial layouts thus reducing the further optimization times, but it is intrinsically faster than the random one. Looking at Table 1, we see that the average execution time for the random method is 48.9 seconds, while for the OP is 39.8 seconds, with a 15% average time saving, statistically significant ($p = 2.2 \times 10^{-10}$ in a paired t-test).

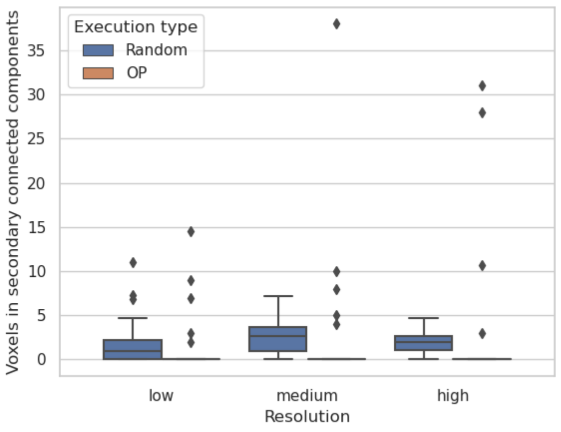

Figure 10: *A box plot representing the average number of voxels in the secondary connected components of the best layouts generated over the course of 10 runs.*

The elapsed time is heavily dependent on the number of voxels, as shown in Figure 12. Computing the averages for the three different resolutions still shows that the OP method is consistently faster than the random one. In this case, the time averages of the OP and random approaches for the low, medium, and high resolution respectively are 0.45, 4.8, and 141.6 (0.39, 4.1, and 114.9) seconds.

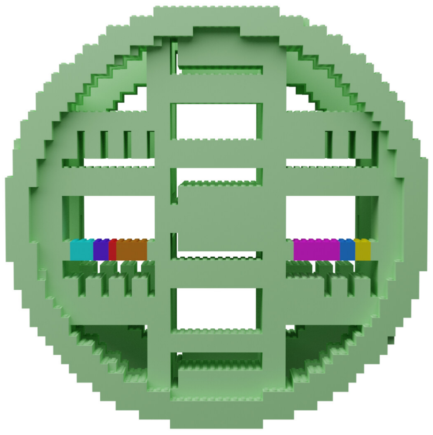

Figure 11: *Rendering of the worst result for the OP method in terms of voxels belonging to secondary connected components. The method fails to connect the thin bridges to the main body, although the optimization process is able to merge them into a single component.*

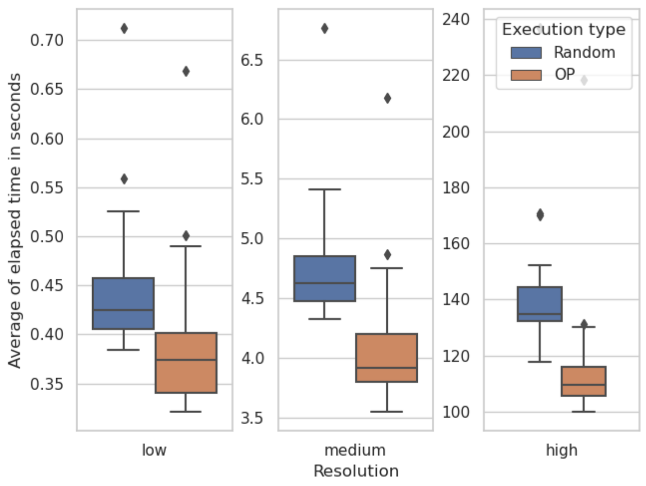

Figure 12: *A box plot representing the elapsed times to create a layout for the different resolutions.*

Furthermore, in Figure 13 we show the average number of bricks used in the layouts created by both methods. We can see how the OP approach is consistently able to use fewer bricks, especially at the higher resolutions. Analyzing only the best layouts generated improves slightly the random results, but it is still far from the numbers of the OP method, as shown in Figure 14.

Lastly, we discuss the sampled node connectivity results. After the building procedure, the node connectivity is 3.56 and 3.71, for the random and OP approaches. The difference is statistically significant with $p = .007$. However, when we analyze the different resolutions independently, we find that for low resolution the results are better for the random approach (2.96 and 2.69, $p = 8 \times 10^{-8}$). Considering the medium resolution we lose significance(3.61 and 3.69, $p = .34$), but at the highest resolution available the OP method

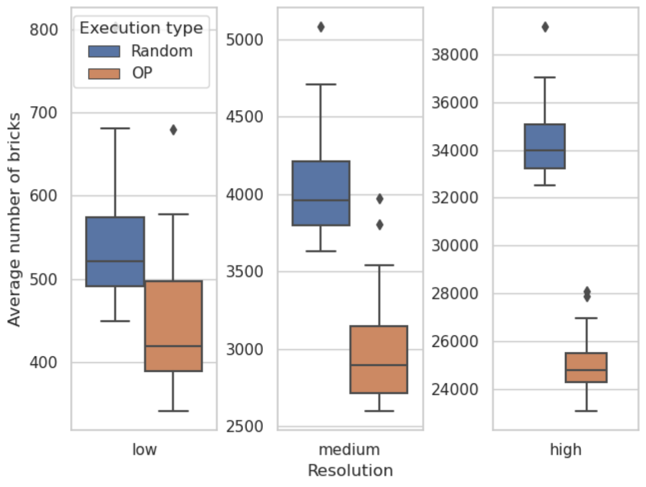

Figure 13: *A box plot representing the average number of bricks used for the layouts.*

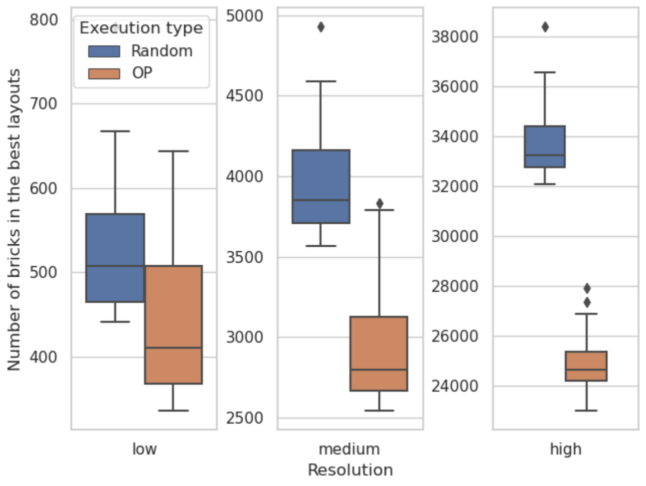

Figure 14: *A box plot representing the number of bricks used for the best layouts.*

produce the best results (4.10 and 4.74, $p = 2 \times 10^{-9}$). Not surprisingly, performing a 2 way ANOVA reveals that the approach, resolution, and their interaction all have $p < .05$.

5.2. Iterative reassembly of the layouts

The OP method is thus able to create better layouts in terms of the number of bricks and connected components, and do this while also being faster than the random method. To prove that this advantage still carries over when an iterative improvement scheme is applied, we measured the same metrics after the execution of the iterative reassembly described in Section 3 on the results of the layout initialization tests described in Section 5.1. We report the result in Table 3.

We limited our test to the iterative reassembly of the regions around boundaries joining disconnected regions until the connected

	Starting layout	Time	Avg bricks	Avg. CC	Average SNC
Lo-res	random	0.89 s	523.3	1.0	3.03
	OP	**0.28 s**	**444.3**	1.0	**2.71**
Mid-res	random	12.5 s	3890.3	**1.0**	3.65
	OP	**3.1 s**	**2972.3**	1.05	**3.70**
Hi-res	random	274.2 s	33075.4	1.04	4.18
	OP	**27.3 s**	**24997.9**	1.04	**4.75**

Table 3: *Efficiency and quality of the results (averaged for the different resolutions and methods) obtained starting from the initialization tests described in Section 5.1 after a subsequent iterative reassembly.*

result is achieved or the algorithm reaches a maximum number of iterations (in all the results reported in this paper, we set it to 20 more the number of connected components).

Our goal is to show that the OP approach is faster and provides a better solution. Clearly, in both cases, the result can be refined further by performing an iterative optimization using any kind of cost function.

Given the different initial numbers of connected components for the two approaches, it is not surprising that the average number of iterations executed is significantly different (6.8 for random initialization versus 1.6 for OP, with the last method providing 80% fewer iterations), resulting in a faster convergence to a single connected component (95.9 versus 10.2 seconds, 84% faster). A more complete graph with the distributions of the average time necessary to run the optimization is shown in Figure 15.

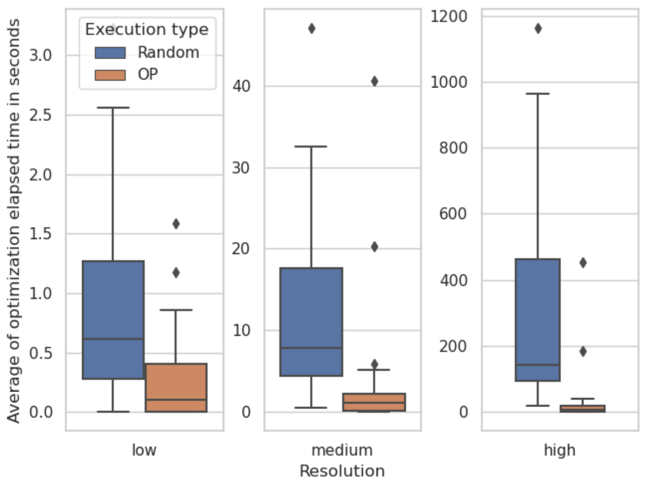

Figure 15: *A box plot representing the average time spent by the optimization procedure.*

It is interesting to note that the single connected component is obtained in almost all the cases, with a negligible amount of exceptions.

Analyzing the results, it is possible to see that all the voxelizations but one can be filled with single connected components of attached LEGO bricks. In very few cases the optimization fails to obtain the connected results, while for a single model this result is impossible.

The single model that is not buildable with standard LEGO bricks and inevitably causes the optimization to fail, regardless of the initialization, is the *dragonstand* model on the high resolution.

The impossible configuration of bricks causing this problem is highlighted in Figure 16, in which two voxels are only adjacent to a third one, but do not form a valid LEGO brick if all are merged.

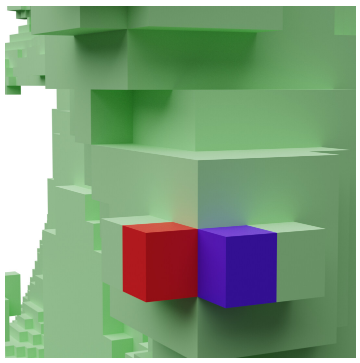

Figure 16: *Close-up of the impossible configuration in the* dragonstand *model. The two highlighted voxels compete for the same support.*

The iterative fix of the connectivity clearly does not change relevantly the brick number, so that the difference between the OP and the random initialization remains the same. This means that, in order to obtain a similar number of bricks with the random method, a further optimization step with the related time complexity should be applied. The distribution of the number of bricks is shown in Figure 17, where it is clear that even after the optimization the layouts derived by the random layouts can not match the number of bricks used by the OP method, and the difference is quite large and statistically significant.

Finally, we discuss briefly the effects on the sampled node connectivity of the graphs.

After the improvement algorithm is applied, the average of the node connectivity seems to slightly favor the heuristic starting layout, but $p = .07$. Furthermore, when applying a 2 way ANOVA to the data, we find that starting from the random or heuristic layout has a statistically significant effect only when combined with the resolution.

These results show that the OP approach gives large advantages with respect to the random initialization. Even starting with several

© 2022 The Author(s)
Eurographics Proceedings © 2022 The Eurographics Association.

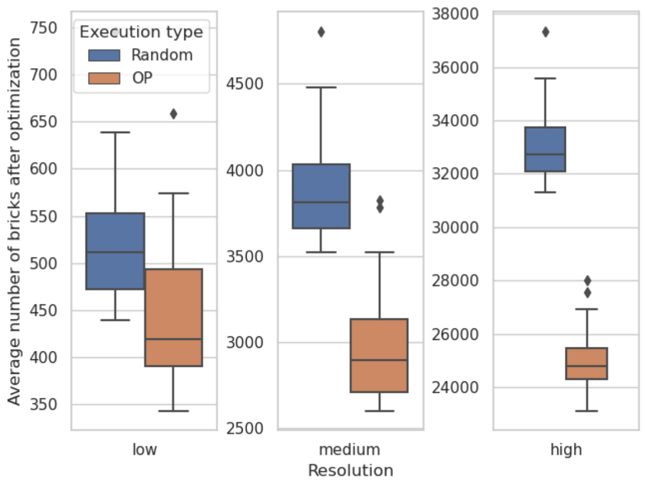

Figure 17: *A box plot representing the average number of bricks used for the layouts after the optimization is performed.*

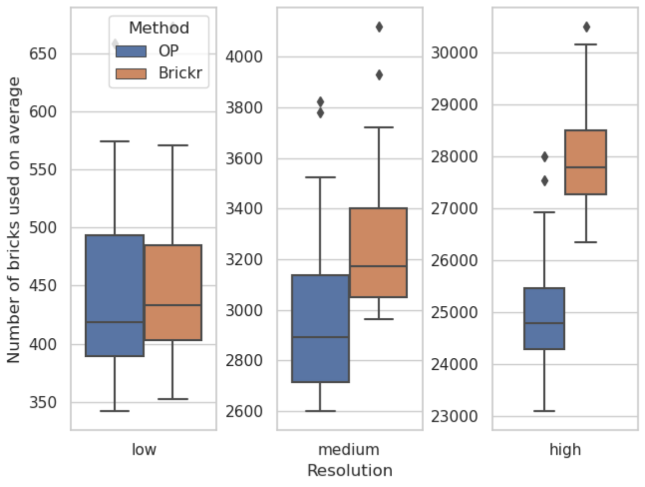

Figure 18: *A box plot representing the average number of bricks used for the layouts by our OP method and by Brickr [TSP13].*

random configurations and considering the best results, the number of blocks will result larger than the one obtained with the OP strategy, and the time complexity higher. And the cost of the creation of several random layouts for large models is quite high, as shown in Table 1.

5.3. Comparison with alternative methods

In order to further evaluate our approach, we tested the Brickr method by [TSP13] on the Bricks dataset. For both methods, we performed ten runs on each model and then averaged the results, which are shown in Table 4. The first thing to notice is that both approaches achieve almost always a single connected component, with a minimal number of exceptions. The differences in the two methods are however noticeable in the times and number of total bricks.

However, it's important to note that our approach is implemented in Python, while Brickr is implemented in C++. As a consequence, any time comparison is inherently flawed, and when considering the two languages' performances [LCSY22], we could even expect our algorithm to perform better than Brickr. On the other hand, the numbers of total bricks are easily comparable, and we see that in the medium and high-resolution models we use on average 10% fewer bricks, as shown in Figure 18.

	Method	Time	Bricks	Conn. comp.
Lo-res	Brickr	0.13s	449.2	1.00
	OP	0.67s	444.3	1.00
Medium	Brickr	0.81s	3269.5	1.00
	OP	7.18s	2972.3	1.05
High	Brickr	26.02s	27963.9	1.03
	OP	142.12s	24997.9	1.04

Table 4: *For each resolution we compare the performances of our OP method versus Brickr [TSP13].*

6. Discussion

The creation of well-connected assemblies of LEGO bricks (or other similar elements) is certainly an extremely challenging problem, with potential applications, not only for artistic purposes, as shown, for example, in [MMG*14]. The problem is not solvable in general and global optimization techniques are not feasible due to the excessive complexity. Many completely different heuristics to build configuration and iteratively increase the connectivity or the physical properties of the results. We suggest that an assembly strategy based on a graph-based brick merging using a globally defined priority privileging the fusion of external and non-attached bricks may provide faster and better results.

To evaluate quantitatively the performances of the heuristics proposed, we also introduced a specifically designed dataset, BRICKS, that allows an objective comparison according to different quality metrics as well as their performances and scalability.

We plan in future work to exploit the dataset to evaluate different kinds of optimization tasks, also related to different connectivity measurements, stability, and forces.

Additionally, an interesting task partially explored by [MMG*14] is the generation of mixed constructions of standard LEGO bricks and 3D printed pieces that conform to the model surface and have holes and studs to attach to the LEGO pieces. This could allow for a more faithful representation of the object at the lower resolutions and is ideally easily obtainable with boolean operators. However, many intersections between the bricks and the surface might be ill-formed and thus require additional merging, and the resulting pieces might be unnecessarily large (thus being slower and more wasteful to 3D print). Taking these constraints into consideration since the layout building could lead to a more efficient and better result.

Acknowledgments This work was partially supported by the MIUR Excellence Departments project 2018-2022.

References

[FDSG21] Fanni, Filippo Andrea, Dal Bello, Alberto, Sbardellini, Simone, and Giachetti, Andrea. "Approximating Shapes with Standard and Custom 3D Printed LEGO Bricks". *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. Ed. by Frosini, Patrizio, Giorgi, Daniela, Melzi, Simone, and Rodolà, Emanuele. The Eurographics Association, 2021. isbn: 978-3-03868-165-6. doi: 10.2312/stag.20211487 1.

[GHP98] Gower, Rebecca AH, Heydtmann, Agnes Eileen, and Petersen, Henrik G. "LEGO: Automated Model Construction". *32nd European Study Group with Industry*. Jens Gravesen and Poul Hjorth, Department of Mathematics, DTU. 1998, 81–94 2.

[KS21] Kollsker, Torkil and Stidsen, Thomas JR. "Optimisation and Static Equilibrium of Three-Dimensional LEGO Constructions". *Operations Research Forum*. Vol. 2. 2. Springer. 2021, 1–52 1–3.

[LCSY22] Lion, David, Chiu, Adrian, Stumm, Michael, and Yuan, Ding. "Investigating Managed Language Runtime Performance: Why JavaScript and Python are 8x and 29x slower than C++, yet Java and Go can be Faster?": *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, July 2022, 835–852. isbn: 978-1-939133-29-40. url: https://www.usenix.org/conference/atc22/presentation/lion 10.

[LKKM15] Lee, Sangyeop, Kim, Jinhyun, Kim, Jae Woo, and Moon, Byung-Ro. "Finding an optimal LEGO® brick layout of voxelized 3D object using a genetic algorithm". *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 2015, 1215–1222 1, 2.

[LKM18] Lee, Seung-Mok, Kim, Jae Woo, and Myung, Hyun. "Split-and-merge-based genetic algorithm (sm-ga) for LEGO brick sculpture optimization". *IEEE Access* 6 (2018), 40429–40438 1–3.

[LYH*15] Luo, Sheng-Jie, Yue, Yonghao, Huang, Chun-Kai, et al. "Legolization: optimizing lego designs". *ACM Transactions on Graphics (TOG)* 34.6 (2015), 222 1–4.

[MMG*14] Mueller, Stefanie, Mohr, Tobias, Guenther, Kerstin, et al. "faBrickation: fast 3D printing of functional objects by integrating construction kit building blocks". *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2014, 3827–3834 10.

[PNA*21] Pietroni, Nico, Nuvoli, Stefano, Alderighi, Thomas, et al. "Reliable Feature-Line Driven Quad-Remeshing". *ACM Trans. Graph.* 40.4 (July 2021). issn: 0730-0301. doi: 10.1145/3450626.3459941. url: https://doi.org/10.1145/3450626.3459941 5.

[Ste16] Stephenson, Ben. "A multi-phase search approach to the LEGO construction problem". *Ninth Annual Symposium on Combinatorial Search*. 2016 2, 3.

[TSP13] Testuz, Romain, Schwartzburg, Yuliy, and Pauly, Mark. "Automatic Generation of Constructable Brick Sculptures". *Eurographics 2013 - Short Papers*. Ed. by Otaduy, M.- A. and Sorkine, O. The Eurographics Association, 2013. doi: 10.2312/conf/EG2013/short/081-084 1, 2, 10.

[WSP21] Wang, Ziqi, Song, Peng, and Pauly, Mark. "State of the art on computational design of assemblies with rigid parts". *Computer Graphics Forum*. Vol. 40. 2. Wiley Online Library. 2021, 633–657 2.

© 2022 The Author(s)
Eurographics Proceedings © 2022 The Eurographics Association.